#5

第55回（平成９年後期）全国大会

講 演 論 文 集 （3）

データベースとメディア

- データベース
- 情報検索
- メディアと情報

ネットワーク

- マルチメディア通信と分散処理
- 分散システム運用
- マルチメディア符号化

平成９年９月24日～26日　　於：福岡工業大学

社団法人 情報処理学会
Information Processing Society of Japan

機動性に配慮した小規模ネットワークの構築経験 −(2)名前空間− [1]

4S−6

上田 仁† 大野 浩之‡
† 東京工業大学大学院 情報理工学研究科 数理計算科学専攻
‡ 東京工業大学大学院 情報理工学研究科

1 はじめに

研究室などの小規模組織は組織構成の変更にともない、いままで所属していた組織とは別の組織のもとに移動する場合がある。移動にともないネットワークのドメイン名が変更されると、ネットワークを利用したいくつかのサービスは継続して利用できない。とくにWorld Wide Web(WWW)は広く一般に利用されており、ホームページを用いて多くの組織がインターネットに向け情報を公開している。このためWWWによる情報提供を継続しておこなえることは重要である。本稿ではまずネットワークの物理的な接続先に依存しない、「サービス名」を中心とした名前空間を導入し、次にWWWに注目し、継続して情報提供を可能にするため「永年利用可能なURL」を提案する。そして永年利用可能なURLを実現するbonsaiシステムの設計および実装について報告する。

2 サービス名

本稿ではホームページを「個人または組織が公開する情報の中で最初に参照されることを想定して作られた情報」と定義する。本稿の目的はホームページを継続して公開可能にすることである。

従来、WWWによる情報提供を継続しておこなうためには移動元に移動先へのポインタを残す方法が用いられてきた。しかしこの方法では組織が移動するごとにポインタ情報を残さなければならない問題がある。

WWWによる情報提供を継続しておこなうために、ドメイン名のかわりに、ホームページ名という新たなサービス名を導入する。ホームページ名は各ホームページごとに名付けられ、インターネット上のホームページを一意に区別し組織が移動しても変化しない。WWWの情報提供者はホームページ名を用いてホームページを公開する。WWWの情報参照者はそのホームページ名でホームページを参照する。

3 永年利用可能なURL

WWWにおけるホームページ名の利用を実現するにあたり、既存のWWWクライアントとサーバを利用する。それにより以下の利点が生まれる。

- 情報の参照者は従来と同じユーザインタフェイスで情報を参照できる。
- 情報の公開者は従来と同じ方法で情報を公開できる。
- WWWの管理者は従来と同じ方法でサーバを管理できる。

ホームページ名を既存のWWWクライアントとサーバで利用するために、URLのなかにホームページ名を含ませた「永年利用可能なURL」を導入する。永年利用可能なURLでは既存のURLのホスト名とホームページの位置をホームページ名に置き換える(図1)。

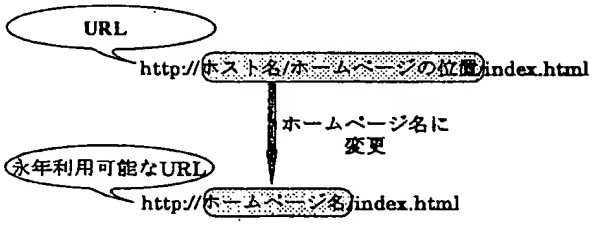


図1: URLと永年利用可能なURLの対応

4 bonsaiシステム

既存のWWWサーバとクライアントを用いて永年利用可能なURLを実現するシステムとして筆者らはbonsaiシステムを開発した。bonsaiシステムではDNSを利用している。DNSはホスト名から計算機のネッ

[1] Networks with Mobility for SOHOs. - Part.1 Name Space. -
Hitoshi UEDA, Hiroyuki OHNO. Department of Mathematical and Computing Science, Graduateschool of Information Science and Engineering, Tokyo Institute of Technology.

トワーク上の位置を知るサービスであるが、bonsaiシステムではホームページ名とホスト名の対応に利用する。具体的には、ホスト名の名前空間の部分空間としてホームページ名の名前空間を構築し、ホスト名に別名をつけるCNAME機能により対応をつけた。DNSによりホームページ名はホスト名に変換され、さらにネットワーク上の位置を示すIPアドレスに変換される。bonsaiシステムは、振り分けサーバ(振り分け部)とCNAME変更デーモン(ネームサーバ管理部)の二つから構成される。

bonsaiシステムによるHTTPの処理方法を図2に示す。HTTPリクエストを送ろうとするWWWクライアントは、DNSを利用してHTTPリクエストを送る先の計算機のネットワーク上の位置を決定する。つぎにその送り先にHTTPリクエストを送る。送り先にはhttpdではなく振り分けサーバが稼働しており、HTTPリクエスト中のホームページ名をWWWサーバのホスト名とホームページの位置に書き換えてWWWサーバに送る。WWWサーバからの応答は振り分けサーバを介してWWWクライアントに中継される。

情報公開者はホームページを継続して公開するため、ホームページが移動したときは移動先の新しい振り分けサーバにホームページ名が対応するように、CNAMEによる対応づけを変更しなければならない。これはCNAME変更デーモンを用いる。CNAME変更デーモンは情報公開者からの電子メールを受け取り、DNSの対応づけを書き換える。このため情報公開者は容易にDNSの対応づけを変更できる。

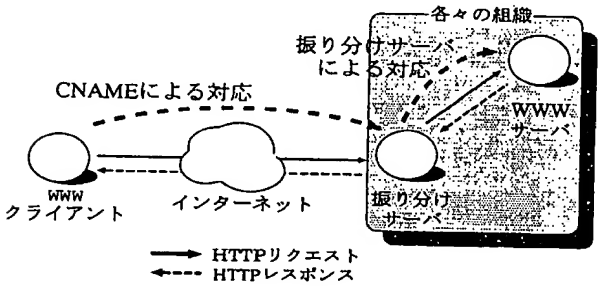


図 2: HTTPの処理

5　bonsaiシステムの実験と課題

bonsaiシステムを用ちいて永年利用可能なURLの実験をおこなった。まず、振り分けサーバとCNAME変更デーモンをperl version 4を用いて実装した。つぎにドメイン名の名前空間の一部分としてwwwhps.is.titech.ac.jpドメインというホームページ名の名前空間をこの実験のために新たにつくり、その下に各個人のホームページ名を登録した。その一つはueda.wwwhps.is.titech.ac.jpである。そして振り分けサーバが動作する計算機とホームページ名をDNSの機能をもちいて対応づけた。WWWサーバはNCSA/1.4.2、WWWクライアントはnetscape-3.01を用いた。

この結果、例えばueda.wwwhps.is.titech.ac.jpというホームページ名で、実際のホームページwww.is.titech.ac.jp/labs/ohnolab/ueda/index.htmlが参照でき、永年利用可能なURLが機能することが確認できた。また、振り分けサーバの対応付けを変えることでホームページが移動してドメイン名が変ってもホームページ名で情報を参照できた。

今後の課題としては、多数のホームページ名を登録するときの名前付けの問題があげられる。ホームページ名は名前空間のなかで一意でなければならない。また、運用規模を大きくして実験することも重要である。

6　まとめ

本稿では小規模組織の移動におけるサービスの継続について述べた。WWWにおいて新たなサービス名を定義し、従来のWWWで利用するために永年利用可能なURLを提案した。永年利用可能なURLを実現するbonsaiシステムを設計し実験をおこなった。

関連研究としては、ドメインが変ってもメールアドレスに影響がないようにする生涯メールアドレスの研究[3]があげられる。また、Top Level Domain(TDL)に関する議論もさかんに行なわれており、1997年2月には7つのTLD(firm,store,web,arts, rec,info,nom)がIAHCにより新たに定義されている[4]。

参考文献

[1] 上田 仁, 永年利用可能なURLの実現手法, 東京工業大学1996年度卒業論文, February 1997.

[2] *WIDE*プロジェクト, 1996年度 WIDEプロジェクト研究報告書(第3部5.2節 永年利用可能なURLの実現手法), 1997., p.85-91

[3] *WIDE*プロジェクト, 1996年度 WIDEプロジェクト研究報告書(第3部5.1節 生涯に渡って利用できる名前空間), 1997., p.81-85

[4] *International AdHoc Committee*, Recommendations for Administration and Management of gTLDs, http://www.iahc.org/draft-iahc-recommend-00.html, 1997.

平成九年九月二四日　発行

編集兼発行者　飯塚浩司
東京都港区芝浦二―六―二〇（芝浦前川ビル7F）

印刷所　三協印刷株式会社
東京都目黒区目黒本町五―二―七

発行所　社団法人情報処理学会
東京都港区芝浦三―一一六―二〇（芝浦前川ビル7F）

第55回（平成９年後期）全国大会講演論文集（3）

平成2年3月15日第三種郵便物認可 平成9年10月1日発行 毎月1回1日発行 第9巻第10号（通巻第103号） ISSN 09...

NTT技術ジャーナル

NTTテレコミュニケーション情報誌

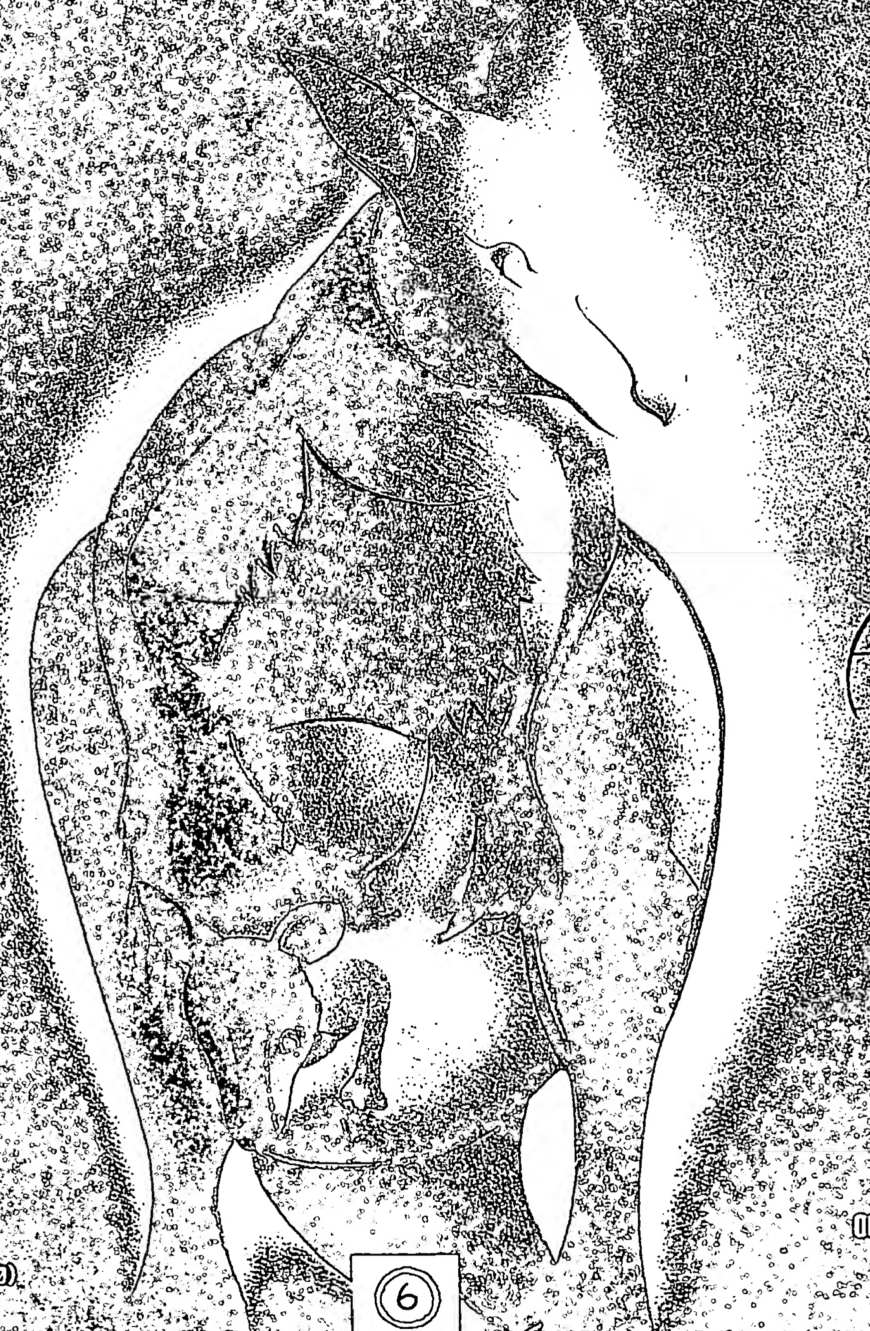
番号機能高度化への取組みについて

〔特集〕番号計画の...

光が見えた!マルチメディアスーパーハイウェーの地下空間

〔特集〕基盤設備の...

●インタビュー 国民の期待にこたえていく それが新生NTTの使命
木塚修一 常務取締役 再編成室長
●CD-ROM付き特別企画 NTTの提供するトータルソリューション
●サービス&システム
ISDN時代に対応するディスプレイホン/発ID対応CTIアプリケーション/ナンバーディスプレイ...
/ナンバーディスプレイ対応ホームファクスの商品化/インターネット上のURL番号...
●トピックレポート/R&Dレポート
様々な音声表現を実現できる音声デザインツールSpeed...
SCC電子商取引実証実験を開始/FAX気象予報サービスの試行実験/インターネットを利用...
実/インターネット系サービス用QoS



OCT. 1997 Vol.9 No.10

（表紙：コミュニケーションする生物）

Service & System

インターネット上のURL番号（ホットコード）変換サーバ

阪本 秀樹（さかもと ひでき）†1 ／野中 優（のなか まさる）†2 ／桝田 吉寛（ますだ よしひろ）†3 ／清水 明宏（しみず あきひろ）†4

NTTでは，インターネットTV等のキーボードを持たないインターネット端末において，見たいURL（Universal Resource Location）の指定を容易にすることを目的とし，インターネット上で“ホットコード”と呼ばれる数字列をそれに対応するURLに変換するサーバを開発しました．このURL変換サーバを利用することにより，端末からは数字列（ホットコード）を入力するだけで，目的のホームページを見ることができるようになります．

はじめに

日本におけるインターネットユーザ数はすでに700万人（日本の人口の6.6%弱）にも達し，そのユーザ層にも変化が現れようとしています．その一例が，昨年末より発売開始されたインターネットTVに代表されるインターネット家電の登場です．インターネット家電は専門的な知識のない初心者でもリモコン操作のみで手軽に操作できることから，一般家庭ユーザ層への普及が期待されています．その一方で，キーボードを持たないため操作が限定される点も指摘されています．NTT（OCN事業部，ヒューマンインタフェース研究所）では，インターネットの家庭での利用の普及促進活動の一環として，キーボードを持たないインターネット家電において，簡単にURLを指定するためのツールであるURL変換サーバの開発を行いました．

URL変換サーバ

■URL変換サーバとは

見たいホームページを直接指定する方法として，通常パソコンでは，URLと呼ばれる文字列をキーボードから入力しています．URLには，例えば，
http://www.nippon.co.jp/~next/TV_2/top.html
のように多くの見慣れない特殊コードを含んでいるため，画面上のソフトキーボードのような簡単な入力機能しか持たないインターネット家電からの入力は極めて困難です．そこで，各URLに対してホットコードと呼ばれる数桁の数字列をあらかじめ割り当てておき，リモコンからチャネル選択の番号ボタン等を使ってURLに対応したホットコードを入力することにより，見たいホームページへジャンプさせるための番号変換を行うのがURL変換サーバです．URL番号変換のための端末とサーバとのネットワーク上のやり取りは，すべて標準的なHTTPプロトコル*1に準じて行われますので，端末側には特殊なプラグインソフトを用意する必要はなく，また，端末機種やブラウザソフトへの依存もありません．実際にクライアントとしては，Windows95上のNetscape Navigator，Internet Explorerの両方での動作を確認しています．なお，変換サーバをネットワーク上に配置したのは，①ホームページは絶えず増加していることから番号テーブルも絶えず更新が必要，②インターネット家電は一般に大容量の蓄積デバイスを持っていない，等の理由からです．

■URL変換サーバの代表的な利用シーケンス

URL変換サーバの代表的な利用シーケンスを図1に示します．

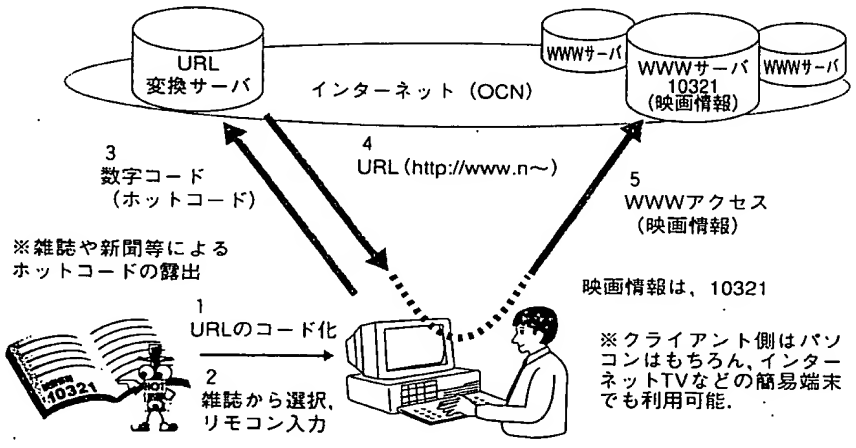


図1　URL変換サーバの代表的な利用シーケンス

*1　HTTPプロトコル：HTMなどの文書や各種マルチメディア情報を転送するためのプロトコル．非常に単純なプロトコルのため各種データや他のプロトコルの伝送も可能なように拡張されたアイコンや文字列をマウスクリックすることで，リンクしたファイルをそのファイルのあるサーバからクライアントへ転送する．WWWで使用されています．

†1　OCN事業部担当課長
†2　同　上　担当
†3　同　上　担当
†4　ヒューマンインタフェース研究所
映像処理研究部主幹研究員

(1) 各URLに対してあらかじめ割り振られたホットコードと呼ばれる数字列は，ホームページを紹介した本（ホームページ帳），雑誌，新聞広告，カタログ，名刺等の紙媒体を通じてユーザに周知されます．

(2) ユーザは，紙媒体から自分が見たいホームページのホットコードを選択し，リモコン等を使って端末から入力します．

(3) ホットコードが端末からURL変換サーバに送られます．

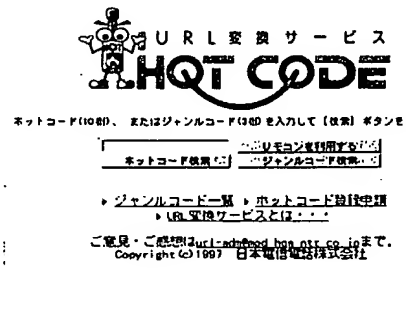
(4) 変換されたURLを含むスクリプトが変換サーバから端末に戻されます．このとき，変換サーバはスクリプト内にURL以外の簡単なメッセージ（広告等）を添付することも可能です．

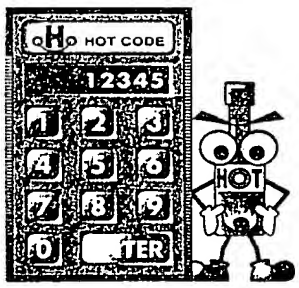
(5) 端末は，そのURLを使ってホームページ検索を行います．このとき，URLを使用した検索はスクリプトに従ってブラウザによって自動的に実行されるため，ユーザの操作は必要としません．

このシーケンスをワークステーション（WS）とパーソナルコンピュータ（PC）を使ったプロトタイプを使って検証しました．プロトタイプでは，URL変換サーバはSun上のCGIプログラムで実現しています．プログラムはperl[*2]で書かれているため移植性がよく，Sunのほかに，WindowsNTや95での動作も確認しています．また，Sun上で，オラクルデータベースをバックエンド結合したRDB[*3]を使った構成での動作も確認しています．

■端末におけるユーザインタフェース

端末におけるユーザインタフェースを図2に示します．端末から，URL変換サーバのページ上のフォームに対してホットコードを入力することにより，サーバにホットコードを通知します．または，別のインタフェースとして，数字入力に必要なフォーム部分のみをリモコン的に別ウインドウ表示し，毎回





●リモコンを使用してコードを入力

「リモコンを利用する」ボタンをクリックすると，コード入力用のリモコンが画面に表示されます．ホームページを表示しているときも，常に画面上に表示することができ，テレビのチャンネル感覚で，いろいろなホームページをすばやく切り替えることができます．

図2 端末におけるユーザインタフェース

インターネット家電とイージーインターネット協会

昨年より，世界中でTV，ワープロ，ゲーム機，カラオケ等のようなパーソナルコンピュータ（PC）以外の機器にインターネットアクセス機能を組み込んだ，いわゆる「インターネット家電」と呼ばれるインターネット端末の出荷が開始されました．これは，今までPCになじみの薄かった一般の初心者層をターゲットとした端末製品であり，電源オンで直ちにブラウザが起動される，リモコンのみでブラウザが操作可能であるなど，使いやすさの点で多くの工夫が見られます．しかし，PCのように多くの機種で同じプラットフォーム，同じOSが採用されていないことから，仕様がばらばらであり，インターネットへの網接続サービスを提供する側，コンテンツを作成・提供する側に混乱をきたしています．そこで，それらの仕様を業界全体で標準化し，インターネット家電の普及促進を図ることを目的とし，昨年10月にイージーインターネット協会（http:// www.eia.or.jp/）が設立されました．現在，協会には家電メーカを中心に，インターネットプロバイダやコンテンツプロバイダ等約60社が加入しており，以下のような議題の審議が行われています．

- インターネットプロバイダの選択，接続パラメータ自動設定の標準手順
- インターネット家電向けコンテンツ作成ガイドライン
- インターネット端末属性表示の標準化
- 組込み機器用機能拡張言語仕様（JavaLite）

*2 perl：主に文字列処理，ファイル処理，プロセス処理などを得意とするスクリプト言語．

*3 RDB；Relational Database：行と列からなる2次元の表でデータを表現する関係データ・モデルに基づいたデータベース構造．関係データベースともいう．複数の表を関連づけて処理することからリレーショナル・データベースという．

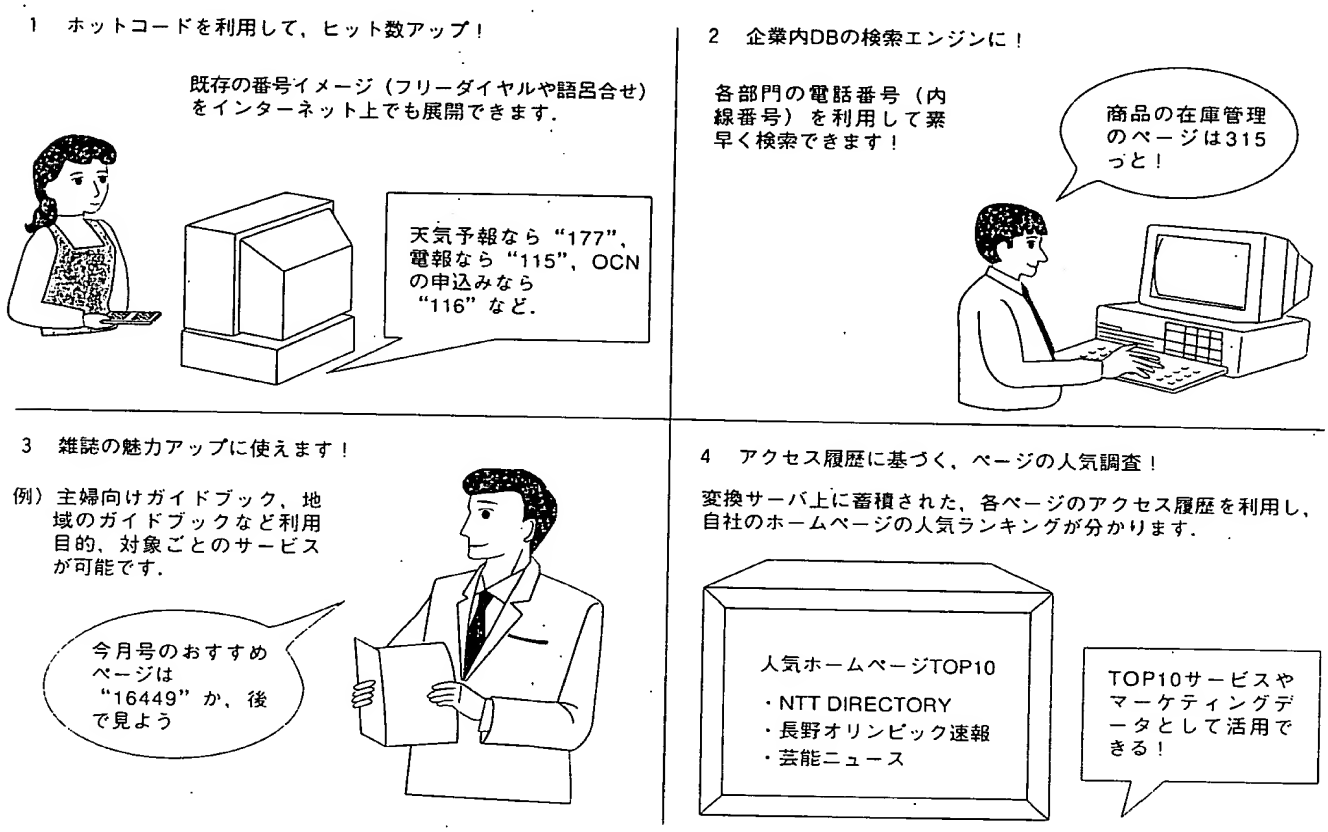


図3　URL変換サーバの応用事例

URL変換サーバの変換のページに戻ることなく，目的ページにジャンプするユーザインタフェースも用意しています．

■URL変換サーバの応用事例

URL変換サーバは，「URL変換サーバの代表的な利用シーケンス」の項で示した電話帳的なURL集によるインターネットナビゲーションのほかに，以下のような個別システムへの応用も可能です（図3）．

(1)　既存番号とのリンクによるナビゲーションシステム

例えば「177で天気予報のページにジャンプする」など，既存の番号イメージを利用し，紙媒体なしでインターネット上の必要なページにジャンプさせるサービス．

(2)　企業内データベースの検索エンジン

イントラネットにおいて，企業内各部門の内線番号やカタログ上の商品番号など数字をキーとして，仕事に必要な情報を取り出す場合の検索エンジンとして利用．

(3)　雑誌の記事等とリンクしたアトラクションツールとして

特定の番号を入力すると，時間や状況の変化に応じて適切なホームページにリンクしてくれるサービス（今，空いているホテルを教えるようなサービス）．

(4)　アクセス履歴に基づくマーケティング情報収集

ユーザのアクセスが必ずいったんURL変換サーバを経由することから，URL変換サーバ上で各ホームページの人気測定等の統計情報収集が可能．

今後の予定

インターネット家電のようなキーボードを持たない端末に対し，簡単にURLを指定するためのツールとして開発したURL変換サーバの紹介をしました．現在，プロトタイプを使用して技術検証を完了したフェーズであり，実際のサービスを開始するためには，「各URLへ割り当てる番号をどのように決めていくか」，「サーバの維持に必要なコストをどのように回収していくのか」，というような運用面での問題点を解決していく必要があります．URL番号体系の統一問題については，インターネット家電のための業界標準化機関であるイージーインターネット協会でも話し合いが始められようとしています．NTTでは，OCNの提供と併せ，今後とも家庭でのインターネットのより使いやすい環境づくりを目指し，サービスの検討を行っていく予定です．

雑誌コード 02057-10

NTT技術ジャーナル
平成2年3月5日第三種郵便物認可
平成9年10月1日発行(毎月1回1日発行) 第9巻第10号
企画編集／日本電信電話㈱研究開発推進部 〒163-19(非用郵便番号) 東京都新宿区西新宿3-19-2 TEL 03-5359-4230
発行／㈱電気通信協会 〒163-14 東京都新宿区西新宿3-20-2 東京オペラシティタワー32階 TEL 03-5353-0183
定価七四〇円(本体七〇五円)

CGIで拓くWebの新世界

第3回 相互データ交換を実現する

かなた まさかつ

前回はCGIスクリプトからのデータ出力法を取り上げたが、今回はその逆方向の流れを解説する。つまり、受け取ったデータをどのようにCGIで処理していくのか、その方法について説明していこうと思う。

前回のおさらい

ちょっと前回のおさらいをしてみよう。ユーザーからデータを受け取るには、

- 標準入力(POST)
- 環境変数(GET)

の主に2つの方法があることは、第1回でもすでに触れたとおりである(正確にはもう1つコマンド引数を使う方法もある)。ユーザーから渡されるデータは、<FORM>タグのMETHOD属性がPOSTである場合は標準入力から、METHOD属性がGETである場合や<A>タグのHREF属性に直接データを記述した場合は環境変数QUERY_STRINGから受け取れる。どちらの方法で受け取る場合も送られてくるデータは通常1行に収められているので、次のようにしてデータを受け取ればよい。

```
・$data = <STDIN>;                    # POSTの場合
・$data = $ENV{'QUERY_STRING'};       #GETの場合
```

データ受信をしてみよう

おさらいを終えたところで、まずは、リスト1のHTML文を入力してほしい。リスト1は、<FORM>タグによって名前、趣味、特技をユーザーに聞き、内容をCGIスクリプトに送信するものである。

当然、CGIスクリプトがなければ使い物にならないので、取りあえずリスト2のCGIスクリプトをインストールしよう。リスト2は、第1回で紹介した「おうむ返しCGIスクリプト」とほぼ同じものだが、POSTではなくGETで送られたデータを表示するようになっている点が異なる。リスト2を入力し、実行権を与えてCGIスクリプトが実行できるパスに置く。次にリスト1の4行目を、リスト2でインストールしたパスに合わせて書き換えてほしい。

それではさっそくリスト1をWebブラウザから呼び出してみよう(図1)。試しに、第1入力フィールドに「kirin」、第2入力フィールドに「basketball」、第3入力フィールドに「takeuma」と入力し、[Submit Query]ボタンを押してみよう。ここで図2の画面が表示されれば成功である。

データの重箱構造

図2で表示された文字列を見てみると、

リスト1 名前や趣味を聞くHTML

```
<HTML>
<HEAD><TITLE>CGI Test</TITLE></HEAD>
<BODY>
<FORM ACTION="/cgi-bin/get_name.cgi" METHOD="GET"> # この行は適当に変える
    名前:<INPUT TYPE="TEXT" NAME="name"><BR>
    趣味:<INPUT TYPE="TEXT" NAME="hobby"><BR>
    特技:<INPUT TYPE="TEXT" NAME="talent"><BR>
    <INPUT TYPE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

リスト2 データ送信CGI

```
#!/usr/local/bin/perl

print "Content-Type: text/plain\n\n";
print $ENV{'QUERY_STRING'};
```

```
name=kirin&hobby=basketball&talent=takeuma
```

となっている。この文字列は、ユーザーからCGIが受け取ったデータである。

見れば大体の察しはつくであろうが、CGIが受け取ったデータは、図3のようなフォーマットで1行にまとめられている。このフォーマットは、個人的にはとてもよくできていると思っている。なぜなら、＝を接続詞の“は”、＆を“そして”と読めば、そのまま意味が読み取れるからである。たとえば先ほどの、

図1　リスト1の実行結果

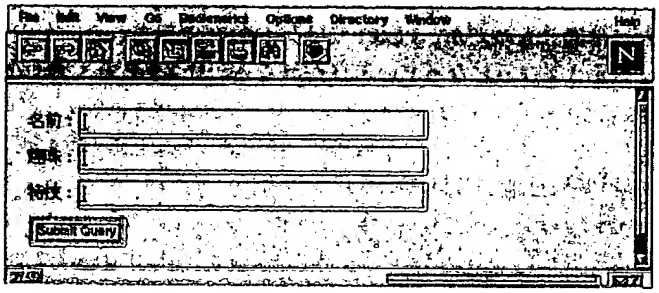


図2　図1の実行結果

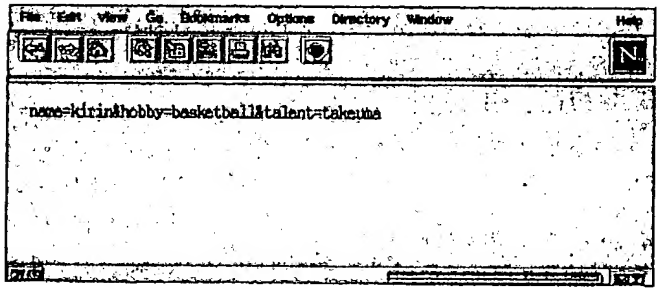
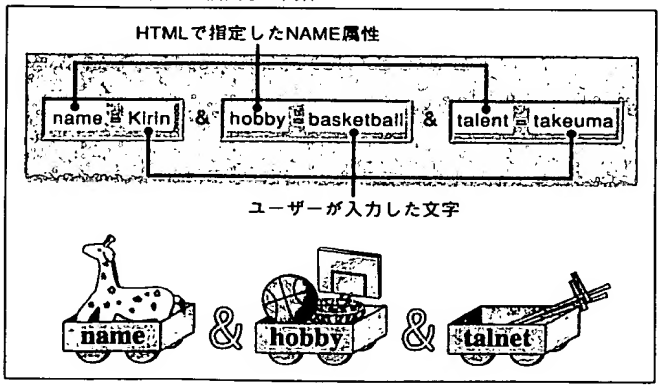


図3　NAME属性と送信文字の関係




```
name=kirin&hobby=basketball&talent=takeuma
```

は、

```
nameはkirin、そしてhobbyはbasketball、そしてtalentはtakeuma
```

と読み取ることができる。

しかし、可読性が高いといっても、CGIスクリプトが読めなければ意味がない。当然、Perlが理解できるように、この文字列を切り分ける必要がある。

Perlで文字列を切り分ける

CGIが受け取った文字列を切り分けるにはいろいろな方法が考えられるが、PerlによるCGIスクリプトでは、NAME属性の値をキーに連想配列を使う方法が一般的なようである。つまり、$query{'name'}とすると、'kirin'が得られるようにデータを振り分けるのである。

仮に、$queryにユーザーから受け取った文字列が入っているとすると、リスト3のようにすることで連想配列%paramに、切り分けた結果が得られる。

自動あいさつCGIスクリプト

データの切り分け方が理解できたところで、ユーザーから受け取った情報を整形して送り出すCGIスクリプトを実際に書いてみよう(リスト4)。リスト4では、リスト3で解説した文字列を切り分ける部分を関数化しているが、働きはまったく同じである。

なお、スクリプト中に日本語が入っているが、JISやシフトJISの場合、日本語コードの中にPerlが解釈してしまう文字列($や@など)を含む場合があるので、Perlスクリプトの中ではEUCコードを使うか、日本語対応のjperlを使うようにしよう。

では先ほどと同様に、リスト4をインストールし、リスト1の4行目を、リスト4をインストールしたパスに合わせて書き換え、実際に実行してみよう。

リスト3　連想配列による文字列の切り分け例

```
foreach(split('&',$query)){          # &で大きく分割
    ($key,$data) = split('=',$_);    # 分割した各要素をさらに=で2分割
    $param{$key} = $data;            # NAME属性値をキーにして入力された文字列を代入
}
```

図4のようになれば成功である。もちろんリスト4の文章は好きに変えてみてもいい。これをうまく改造すれば、卒業論文自動生成CGIなんてのもできるかもしれない。

記号、日本語を受け取るには

リスト4でしばらく遊んでみた読者は気づいたかもしれないが、実は、リスト4では日本語や一部の記号は表示されないのである(図5)。

なぜ日本語や記号が化けるのか

日本語や一部の記号が文字化けするのは、いくつかの理由がある。まず＝や＆といった文字が、受け取ったデータの中に入っている場合だ。たとえば、図1のnameのところに「aa&name=bb」と入力した場合、CGIスクリプトに送られる文字列は、

```
name=aa&name=bb&hobby=.....
```

のようになってしまい、CGIスクリプトはユーザーの名前が"aa"なのか"bb"なのか判断できず混乱してしまう(本当はそのどちらでもないのだが)。

またURLには、コード0x7F～0xFFの文字(これはEUCやシフトJISにしばしば現れる)、制御コード(コード0x00～0x1F、JISが使う0x1Bのエスケープ・コードはこれに含まれる)、そして一部の記号やスペースなどを含めることができない[注1]。したがって、URLにデータを付加して送るGETでは、そのまま日本語や一部の記号などを送ることができない。

そのためWebブラウザは、ユーザーから受け取った文字列中に一部の記号やコード0x7F～0xFFの文字などが含まれていた場合、これらの文字をエスケープ(別の文字で表すこと)してからhttpdに送るようになっている。

Webブラウザはどうやってエスケープするのか

このエスケープの仕組みは簡単である。エスケープしなければならない文字のキャラクタ・コードを2桁の16進数で表し、その先頭に"%"を付けるだけだ。

たとえば、"！"をエスケープするには、「%21」のようにすればよい。日本語の場合も同じである。たとえば、EUCの"あ(0xA4A2)"をエスケープするには、"%A4%A2"のようにすればよい[注2]。また"%"自身はどうなるのかといえば、もちろんこれも"%25"とエスケープすればよい。ただし、スペース(' ')だけは、なぜか"+"でエスケープされるようになって

リスト4 自動あいさつCGIスクリプト

```
#!/usr/local/bin/perl

%param = &get_param; # ユーザーからのデータを得る

print <<"EOT";        # EOT行までを出力する
Content-Type: text/html

<HTML>
<HEAD><TITLE>CGI Test</TITLE></HEAD>
<BODY>
<H1>$param{'name'}さんへ</H1>
<HR>
$param{'name'}さん、はじめまして。<BR>
$param{'hobby'}がお好きだそうですね。<BR>
今度、ぜひ$param{'name'}さんの$param{'talent'}を見せてください。
</BODY></HTML>
EOT

sub get_param{                          # データを切り分けて連想配列を
    local($query,$key,$data,%param);    # 返す関数
    $query = $ENV{'QUERY_STRING'};      # GETの場合のデータを得る
    foreach(split('&',$query)){
        ($key,$data) = split('=',$_);
        $param{$key} = $data;
    }
    %param;                             # 関数get_paramの返り値
}
```

図4　リスト4の実行結果

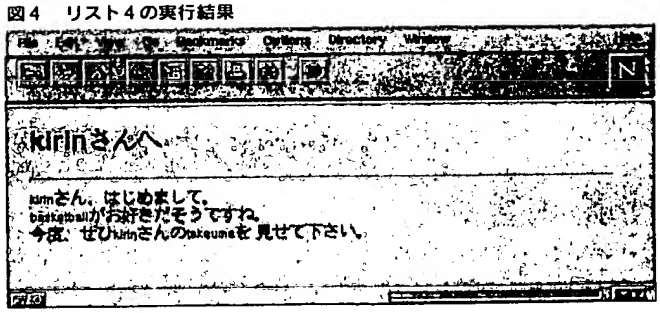
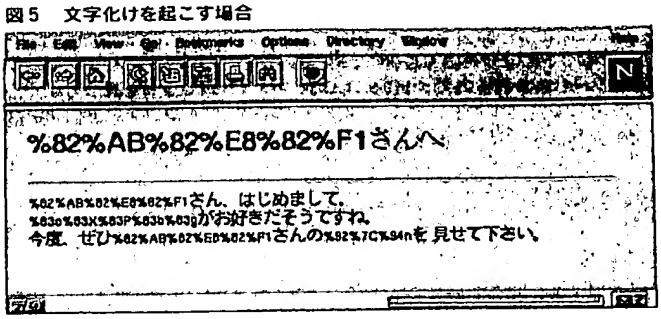


図5　文字化けを起こす場合



注1
URLに含めてはいけない文字の詳細やその理由については、http://ds.internic.net/rfc/rfc1738.txtを参照されたい。＜や＞が危険だとする理由などは、考案者の思慮深さが感じられて面白い。

いるので注意が必要だ。

なおこれは余談だが、最近では/~cgi/といったURLパスを、/%7Ecgi/と書く風潮があるようだ。

エスケープされたコードを復元する

Webブラウザ側で文字列をチェックし自動的にエスケープしてデータを送ってくれることはご理解いただけただろう。ただし、httpdはエスケープの復元までは行ってくれないので、CGIスクリプト側でエスケープされたコードを復元する必要がある。

たとえば、エスケープされた文字列$escapedを元に戻すには、リスト5のようにすればよい。読者の中には恐らく「うわあ、なんじゃこりや」と思った方もあろうが、落ち着いて1行ずつ解読していこう。

1行目は説明しなくとも分かるだろう。＋をスペースに置換しているだけである。問題は2行目である。

リスト5　エスケープされたコードを復元する

```
$escaped =~ s/\+/ /g;
$escaped =~ s/%([\dA-F][\dA-F])/pack("C",hex($1))/egi;
```

注2：16進数中のアルファベットのA～Fは、"%84%82"のように小文字で表記してもよい。

これからの説明はちょっとややこしくなってしまうが、この2行は丸暗記（または丸コピー・アンド・ペースト）してしまえば、理屈が分かっていなくてもエスケープを元に戻すことくらいはできるので、さっぱり訳が分からなくても心配する必要はない。

ただし筆者の経験上、「これはおまじないだから丸暗記」ですまされると、非常にすっきりしない気分になったり、いざというときに非常に困ったりすることが多い。それを避けるために取りあえず説明を加えておくので、難しいことが嫌いな読者は読み飛ばしてほしい。

まずこの式は、s///関数を使った置換を行っているのが分かるだろう。もちろん、エスケープされた文字をエスケープ前のものに置換しているのである。

"s/"の後の"%([\dA-F][\dA-F])"の部分を見てほしい。まず、s///関数は、エスケープ文字列である%で始まって、その次に0～9（\d）かA～F（A-F）のいずれかに当てはまる2文字が続く文字列を探す。つまり、エスケープされた文字にマッチするわけだ。また、s///関数の最後の部分のiオプション（右辺最右端）によって、A～Fの文字は小文字でもよいことになっている。つまり、"%AB"でも"%ab"でもちゃんとマッチすることになる。そして()で"[\da-f][\da-f]"が囲まれているので、文字コードに当たる部分は、特殊変数$1に収められる。たとえば、！のエスケープである%21の場合、

COLUMN　エラーが出たら？

CGIスクリプトのデバッグは厄介である。CGIスクリプトに誤りがあってプログラムが異常終了した場合、httpdはブラウザに「サーバー・エラーなんだけど、たぶん内部エラーか設定ミスじゃないかな」程度のとても大まかな情報しか返してくれない。つまり、何が原因でエラーが起きたのか、ブラウザの表示からはさっぱり分からないのである。

CGIのエラー原因を探る基本は、httpdのエラー・ログに記録されるCGIスクリプトのエラー出力を調べることだ。ただし、エラー・ログにはほかのエラーも含まれるので、目的のCGIスクリプトのエラーを見つけにくかったり、CGIスクリプトを書く権利はあってもエラー・ログを見る権利がなかったりする場合がある。

そういう場合には、プログラムの始めに、リストAのようなコードを埋め込めばよい。こうすればエラー出力は、httpdのエラー・ログの代わりに読者の好きなファイルに落とすことができる。ただし、これはPerl 5.xの機能を使っているので、Perl 4.xでは利用できない。

なお、ブラウザやエラー・ログに、「reason: malformed header from script（スクリプトからおかしなヘッダーを受け取った）」などというエラーが書かれている場合は、たいていHTTPヘッダが間違っているか、HTTPヘッダを出力する前に別の文字列を出力してしまったかのどちらかであろう（例1）。また、CGIスクリプトの結果を、ブラウザがダウンロードしようとする場合は、多くの場合はスペル・ミスなどに起因するContent-Type行の設定ミスの可能性が高い。

例1　典型的な誤り

```
○正しい
    print "Content-Type: text/plain\n\n";
    print "Hello!";

×間違い（HTTPヘッダの前に何か表示しようとしている）
    print "hello!\n";
    print "Content-Type: text/plain\n\n";
×間違い（: が ; になっている）
    print "Content-Type; text/plain\n";
    print "Hello!";
×間違い（\nが1つしかない）
    print "Content-Type: text/plain\n";
    print "Hello!";
```

リストA　CGIエラー・ログ出力ルーチン

```
BEGIN{
    $log_file = './Error_log_dir/error.log'; #パスは適宜変更のこと
    close(STDERR);
    open(STDERR,">>$log_file");
}
```

$1には文字列21が収められる。

次に、最後の"/egi"の部分に注目しよう。eオプションは、置換文s/A/B/のBの部分(ここではpack("C",hex($1))に相当)をPerlの式と見なし、Aの部分で指定したパターンにマッチした文字列を式の結果で置き換えることを意味している。

ではBの部分に当たるpack("C",hex($1))の部分を見てみよう。ここではpack()関数の詳細には触れないが、たとえば、pack("C",$code)のようにすれば、$codeを10進数の文字コードとして解釈し、その文字を返すようになっている。これで本来の目的はほぼ達成できるようだが、実は問題がある。それは、$1に入っているキャラクタ・コードが16進数であることだ。

したがって、16進数を10進数に変換してやらなければならない。そのために使用しているのが、hex()関数である。hex()関数は、16進数の文字列を受け取って、10進数の値を返す関数である。たとえば、%21がマッチした場合、hex()関数は10進数の33を返すようになっている。

たとえば、先ほどの%21がマッチした場合、pack("C",hex($1))はpack("C",33)に置き換えられるので、無事文字!が得られることになる。そして最後のgオプションにより、同様の置換が全文に対して行われるのである。

日本語コードの問題

さて、エスケープを復元できたから、もう日本語を入力しても大丈夫、と思った方は早計である。そう、漢字コードの問題だ。Webブラウザ側は、特殊文字はエスケープしてくれるが、漢字コードまでは統一してはくれない。たとえば、多くのPCはいわゆるシフトJISで日本語を送信しようとするが、リスト3に書かれている日本語はEUCになっているはずだ。リスト3をシフトJISで書いたところで、今度はUNIXなどのユーザーがEUCで日本語を送信した場合に、やはり漢字コードが混在して、恐らくWebブラウザは文字化けを起こしてしまう。

本来なら、ここで漢字コードの判別の仕方や変換の仕方を紹介すべきなのだが、漢字コードの判別や変換は難しく、とてもではないがここでは誌面が足りない。というわけで、やむなくフリーで出回っているパッケージのお世話になろう。

jcode.plを利用した文字列の切り分け法

今回は歌代和正氏の作ったjcode.plというパッケージ[注3]を使用する。このパッケージの詳しい使用法は、ソース・コードを参照されたい。このパッケージは、多くのBBSやチャットの日本語処理に使用されているので、一度は目にしたことのある方も多いことだろう。

取りあえず、このjcode.plをリスト4に組み込んでみよう。jcode.plを/usr/local/lib/perl/などPerlライブラリとしてパスが通っているディレクトリにコピーし、リスト3の3行目に、次の行を挿入しよう。

```
require 'jcode.pl';
```

これで、jcode.plパッケージの関数を呼び出すことができる。次にjcode.plのソース・コードを見ると、漢字コードを特定のものに変換するには、

```
&jcode'convert(*input_line,$output_code);
```

とするように書いてある。この呼び出し方を参考にリスト4のget_param関数を書き換えてみよう。なお、*input_lineの部分は、変換元の文字列が入っている変数の$記号を*に変えたものである。リスト4の場合、変換元の文字列は$dataなので、この部分は、*dataとすればよいことになる(リスト6[注4])。なお、リスト6では、連想配列のキーに当たる部分

リスト6　jcode.plを利用した文字列の切り分け

```
# データを切り分けて、連想配列を返す関数
sub get_param{
    local($query,$key,$data,%param);
    $query = $ENV{'QUERY_STRING'};    # GETの場合のデータを得る
    foreach(split('&',$query)){
        ($key,$data) = split('=',$_);
        $data =~ s/\+/ /g;
        $data =~ s/%([\da-f]{\da-f])/pack("C",hex($1))/egi; # まずエスケープを元に戻す
        &jcode'convert(*data,'euc');      # 漢字コードをEUCに変換する
        $param{$key} = $data;
    }
    %param;                               # 関数get_paramの返り値
}
```

注3：
jcode.plは、ftp://ftp.iij-ad.jp/pub/IIJ/dist/utashiro/perl/jcode.pl-2.3などから入手できる。

注4：
Perlで関数を呼び出す場合、通常、値渡しを用いるが、こうすることで参照渡しを実現できるようになる(Cが分かる方はポインタ渡しといえば分かりやすいだろう)。参照渡しを利用することで、変数の内容が関数によって直接書き換えられるのだ。
jcode.plがこのような方法を採ったのは、おそらくメモリ節約のためだと考えられる。

(たとえばname=kirinのnameの部分)は、エスケープの復元や漢字コードの変換は行っていない。

それではさっそく試してみよう。うまく日本語が表示されただろうか(図6)？　なおリスト6の関数は、少しずつ改良を加えながら、今後も使っていく予定であるので、リストをよく読んで理解を深めておいてほしい。

POSTで受け取るには

さて、get_param関数はいまのところ、GETで送られた情報しか受け取ることができない。そこで、POSTで受け取るには、get_paramの3行目を、

```
$query = <STDIN>;
```

のようにすればよいのだが、これでは今度はGETで受け取ることができなくなってしまう。何とか自動的にPOSTかGETか判断する方法はないのだろうか。

実は、データがPOSTで送られたかGETで送られたかは、環境変数REQUEST_METHODに入れられているので、POST、GETのどちらのデータでも受け取れるようにするには、get_param関数をリスト7のように設定すればよいのだ。環境変数REQUEST_METHODには、POSTでデータが送られた場合は文字列"POST"が、GETで送られた場合やデータが送られていない場合は文字列"GET"が収められている。それ以外の値が入っていることはまずない。

ここで一つ注意してもらいたいことに、データの大きさがある。リスト7の4行目を見てほしい。

ユーザーがGETでデータを送信する場合、URLの末尾にデータを追加してhttpdに送るのだが、URLにはもともと文字数の制限があるので、データのサイズが大きすぎるといった心配はない。

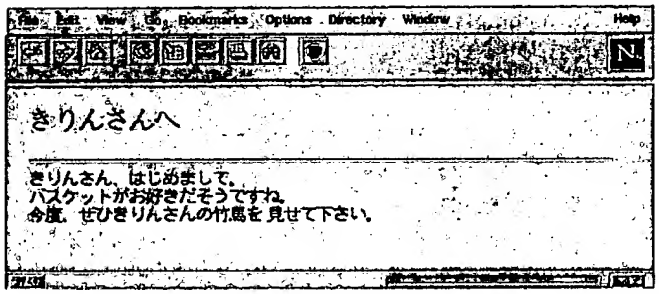
しかし、POSTの場合、何十Mバイトものデータを受け取れるので、メモリを食いつぶさせたり、ゲスト・ブックのように受け取ったデータを保存するタイプのCGIスクリプトの場合、ハードディスクをパンクさせるといったクラックが可能となってしまう。

ここで環境変数CONTENT_LENGTHを使うのだが、この変数にはPOSTで送られてくるデータのバイト数がhttpdによって収められている。この例では、この変数をチェックして、送られてくるデータが1024の3乗バイト、すなわち1Mバイト以上だとCGIスクリプトが異常終了するようになっている。なお、<STDIN>から読み込む文字列は、環境変数CONTENT_LENGTHに入っているバイト数と全く同じなので、「httpdなんて信用できない」といった方や「それでも<STQIN>から大量のデータが送られてくるかもしれない」といった心配性の方は、リスト7の5行目を、

```
$query = read(STDIN,$query,$ENV{'CONTENT_LENGTH'});
```

のようにすればよいだろう。

図6　リスト6の実行結果



リスト7　環境変数REQUEST_METHODによる切り替え

```
sub get_param{
    local($query,$key,$data,%param);
    if    ($ENV{'REQUEST_METHOD'} eq 'POST'){
        die if ($ENV{'CONTENT_LENGTH'} > 1000**3); # 1MB以上のデータは受け取らない
        $query = <STDIN>;                          # POSTの場合
    }elsif($ENV{'REQUEST_METHOD'} eq 'GET'){
        $query = $ENV{'QUERY_STRING'};             # GETの場合
    }else{
        die "Unknown request method";              # 正常なリクエストではない
    }
    foreach(split('&',$query)){
        ($key,$data) = split('=',$_);
    $data =~ s/+/ /g;
    $data =~ s/%([\da-f][\da-f])/pack("C",hex($1))/egi; # エスケープを元に戻す
    &jcode'convert(*data,'euc');                       # 漢字コードをEUCに変換する
       $param{$key} = $data;
    }
    %param;                                        # 関数get_paramの返り値
}
```

次回からはいよいよ本格的なCGIへ

今回までの解説で、CGIスクリプトによるデータの入出力について一通り解説を行った。次回からいよいよ、データの入出力を伴なった実用的なCGIスクリプトの記述法の解説に移る予定である。なお、今回解説した漢字コードやエスケープの問題については、今後もよく出てくるので内容をよく理解しておいてほしい。